



The Productivity Gap

Why Faster Code Doesn't Mean
Faster Delivery

abstract-group.com

Executive Summary

Technology leaders have invested heavily in AI coding tools, [expecting faster code to mean faster delivery](#).

Developers are writing code faster yet lead time for changes, deployment frequency, and time to market have barely moved because coding is a minority activity; so, speeding it up still leaves review queues, test cycles, and approval chains exactly where they were.

This whitepaper examines where delivery time is spent and what happens when output outruns the pipeline, particularly in regulated sectors. It argues that the productivity gap is not a

tooling problem, but a process and governance problem, and the organisations seeing the highest returns from AI in engineering are the ones that have redesigned how their teams work, not simply those with the most advanced tools.

By the end of this paper, the reader will have a three-stage model for assessing AI adoption maturity, a set of diagnostic questions to test whether delivery has genuinely accelerated, and a practical starting point for closing the gap.



Contents

04	The Claim and the Contradiction
05	Where the Time Goes
07	Why the Other Stages Don't Move on Their Own
09	What Happens When Output Outruns the Pipeline
11	The Three Stages of AI Adoption in Engineering
11	Stage 1 – Individual Tool
12	Stage 2 – Team Workflow
12	Stage 3 – Pipeline-Level Automation
13	Questions for Technology Leaders to Ask
15	The Recommendation
17	References

The Claim and the Contradiction

The technology industry has invested heavily in AI-assisted coding tools, expecting a proportional acceleration in software delivery. [The assumption is straightforward](#); if individual developers write code faster, teams will deliver software faster, and organisations will reach market with new capabilities more quickly.

However, the reality is more complicated.

Across technology organisations, a pattern has emerged that challenges this assumption. Developers report that AI coding tools have [improved their code output velocity](#), yet the metrics that matter most to the business (lead time for changes, deployment frequency, time to market, and cost per feature) have not moved proportionally. In many cases, they have not moved at all.

This pattern holds across organisational contexts; over **50%** of developers report that AI has improved their team's output by **10%** or less, with some reporting no impact whatsoever (*Gartner, 2025*). Among the organisations

achieving measurable gains, only the highest performers (the top quintile) report **16 - 30%** improvements in productivity and time to market, alongside **31 - 45%** gains in software quality (*McKinsey & Company, 2025*). For the majority, the productivity gains from code generation tools remain modest or immaterial.

The disconnect creates a pointed question that technology leaders are increasingly asking: [where has the organisation's investment in AI tooling and resulting productivity gains shown up in speed to market, cost reduction, or measurable business value?](#)

The answer to that question requires looking beyond code velocity to the architecture of software delivery itself.

Where the Time Goes

The gap between code velocity and delivery speed exists because developers do **not** spend the majority of their time writing code.

Research from Atlassian's 2025 'State of Developer Experience' report shows that developers only spend around **16%** of their time writing code, with the remaining **84%** distributed across design, testing, debugging, meetings, and waiting for other parts of the delivery pipeline to complete (*Atlassian, 2025*).

Earlier research showed that developers spend only **24%** of their time writing code, whilst the remaining **76%** is distributed across the other activities (*CIO.com/Forrester 2024*).

This ratio varies by role and context (architects and business analysts

spend considerably more time in design, testers in testing phases) but the fundamental pattern is consistent, active code composition is a minority activity in the software delivery lifecycle. Atlassian's own researchers agree that **coding was never really the friction point**, which is exactly why coding assistants can improve the experience of writing software without improving the overall efficiency of delivery.



AI coding tools address one stage of a multi-stage process; a **developer who writes a function in half the time, assisted by an AI tool, still waits the same number of days for code review**. The test suite runs for the same duration, the release approval chain takes the same amount of calendar time, and the deployment window operates on the same schedule.

In an examination of developer time allocation across a cohort of organisations adopting coding assistants, developers spent approximately **30 - 40%** of their time actively coding. The remaining **60 - 70%** was allocated to code review, clarifications with team members, waiting for test results, and involvement in approval processes. When AI tools accelerate the **30 - 40%** coding phase, the total productivity gain is necessarily constrained by the **60 - 70%** of time spent in stages the tools do not affect.

The most time-consuming activities remain fixed; a code review queue does not shrink because individual developers write code faster, a test regression suite does not execute more quickly because the code was generated by an AI tool and review delays cannot be reduced **regardless of how rapidly the code was produced**.

In one project engagement, feature delivery velocity improved significantly through the use of AI-assisted development. The team's code output accelerated measurably; however, **the capability of the test environment became the constraining factor**. Engineers could only test their code in series, one at a time so the bottleneck shifted from code generation to test execution. Only when the test environment was rearchitected to allow parallel deployments and multiple engineers testing simultaneously did delivery velocity increase materially. The constraint moved, it did not disappear so the productivity gain from AI tooling was only visible after the downstream bottleneck was addressed.

Why the Other Stages Don't Move on Their Own

The bottlenecks that constrain delivery exist independently of code generation speed, they predate AI tooling and persist regardless of improvements in how quickly developers write code.

Code review capacity is fundamentally constrained by the availability of experienced reviewers. A faster first draft does not reduce the queue of code awaiting review; it increases the volume entering the queue. In organisations where review capacity is already limited, **faster code generation exacerbates the bottleneck** rather than alleviates it. A human reviewer with a finite capacity can quickly become overwhelmed by the volume of code produced by AI systems. The reviewer then faces a choice: invest significant time reviewing every submission, becoming a bottleneck themselves, or approve submissions with inadequate scrutiny, shifting the risk downstream to testing and production.

Test infrastructure and automation are similarly independent of code generation velocity. Test suite runtimes are determined by the breadth and depth of test coverage, the infrastructure available to execute tests in parallel, and the maturity of test automation frameworks. None of these factors change because developers write code faster. If an organisation runs a regression suite that requires twelve hours to complete, that suite will take twelve hours **regardless of whether the code was generated by AI or written manually.**

Release approval chains are organisational and governance structures. They exist to manage risk, ensure compliance, and enforce quality standards. The time these chains require is determined by the number of approvals, the decision-making authority of each approver, and the availability of those approvers. These processes predate AI tooling and operate independently of code generation speed.

Environment availability also constrains which work can progress.

If a test environment is unavailable, or available only at certain hours, or accessible only to one developer at a time, then code cannot be tested **regardless of how quickly it was written**. The same applies to staging environments, pre-production systems, and production deployment windows.

Technology leaders should examine whether delivery has genuinely accelerated since AI adoption. The hard diagnostic question is whether deployment frequency has improved, or whether code is sitting in the same release approval queues and test cycles as before, only arriving faster. Most organisations have not bridged the gap between code generation speed and pipeline automation. The use of AI tools with reusable skills and templated approaches can increase requirements clarity and reduce rework. However, organisations typically treat AI code generation as a tooling decision without equivalent investment in the other stages of the delivery pipeline. **The result is that code generation accelerates, but the pipeline as a whole does not.**

The research on this dynamic is clear. Among top-performing organisations, **57%** invest in hands-on coaching and training to support AI adoption, whilst only **20%** of bottom performers do so (*McKinsey & Company, 2025*). This matters because the stages beyond code generation do not improve simply because a tool has been installed. They improve when teams learn new ways of working, such as writing clearer specifications, reviewing AI-generated code efficiently and embedding reusable skills and templates into everyday practice. Top performers treat AI adoption as a capability and process redesign problem whereas most organisations treat it as a tool distribution problem, and consequently, the expected productivity gains fail to materialise.

What Happens When Output Outruns the Pipeline

Unmanaged acceleration creates new risks; when code generation velocity increases faster than review, testing, and approval capacity, organisations ship at higher rates without equivalent improvements in governance or quality assurance. The consequences are particularly serious in regulated sectors.

The concern is multifaceted, AI-generated code also introduces hidden defects and creates false

confidence in security. Research into developer experience with AI tooling found 53% of developers report that AI generated code often appears correct yet introduces hidden defects (*Sonar, 2026*). The verification burden then falls on reviewers and testers and if review capacity is already constrained, the risk of defects passing into production increases.



The business impact differs significantly by sector. In financial services, insurance, legal services, and utilities, [a change failure does not result in a straightforward rollback](#); regulatory consequences, customer liability, and operational disruption follow. The cost of a defect in these environments is measured in regulatory fines, customer compensation, and reputational damage, not simply in engineering effort to fix the issue.

Beyond individual defects, the accumulation of faster-coded solutions without equivalent governance creates technical debt. When multiple teams accelerate code delivery without coherent architectural governance, shared technology standards, or a process for decommissioning legacy solutions, the organisation accumulates layers of incompletely integrated systems. Each piece solves an immediate problem rapidly, but the portfolio becomes increasingly complex to maintain, increasingly difficult to modify, and increasingly expensive to operate. [Technical complexity compounds and organisational capacity to change is diminished](#). The research evidence on this dynamic is stark. Two-thirds of organisations report efficiency and productivity gains from AI, yet only **20%** have achieved revenue growth from their AI initiatives (*Deloitte, 2026*).

Furthermore, the most recent DORA (DevOps Research and Assessment) research, the industry standard measure of software delivery performance, found that [AI adoption has turned from a negative influence on throughput into a positive one](#) compared with the previous year's findings, but delivery stability continues to decline as adoption rises, because AI is functioning as an amplifier, strengthening organisations that already have solid engineering foundations in place and exposing those that do not (*DORA, 2025*).

The challenges created by unmanaged acceleration have shifted from productivity to stability, and in regulated sectors, stability is where risk has the greatest impact.

The Three Stages of AI Adoption in Engineering

The path from AI as individual tool to AI-augmented delivery requires progression through distinct stages, each with different operational characteristics and ROI implications.

1

Stage 1 - Individual Tool: A developer using a coding assistant (GitHub Copilot, Cursor, or similar) as part of their individual workflow will find that the tool increases their code output velocity but **it does not change how the team works**, how code is reviewed, how testing proceeds, or how work is deployed. A developer who adopts a coding assistant generates code faster, but the rest of the delivery pipeline remains unchanged. If the organisation removed the AI coding tool tomorrow, the individual developer's velocity would decrease, but the team's delivery metrics would remain materially unchanged. **The tool has optimised one stage of the pipeline without touching the others.**

In organisations that remain at this stage, the productivity gains from AI tooling are constrained by downstream bottlenecks. For example, one early adoption of GitHub Copilot across a cohort of developers showed mixed results. Some engineers gained value; others reported little or even an impediment to their productivity. On reflection, the organisation should have taken a baseline of DORA software delivery metrics before adoption, then measured the actual impact on those metrics. **Without such measurement, the value of the investment remains unclear.** The adoption was treated as a technology decision rather than as part of a deliberate, measured business transformation.

2

Stage 2 - Team Workflow: Shared standards on how AI tools are used emerge, and the team's processes begin to adapt around them. AI-generated code has different quality patterns from manually written code meaning code review processes and testing strategies adapt. As tooling and methodologies converge across roles, **AI extends beyond software engineering**, it supports business analysts in analysing requirements and testers in developing test cases, complementing the structured approaches and frameworks already used in each discipline. As a result, **all stages of the delivery pipeline apply AI where it creates value**. When organisations implement this approach thoughtfully, delivery velocity improves measurably, typically 2x with the same team sizes.

3

Stage 3 - Pipeline-Level Automation: Fully agentic systems coordinate review, testing, security assessment, deployment, and monitoring, not just code generation so **a working MVP can be delivered in two months rather than six**. Teams become smaller and can operate with more junior staff, because AI support augments their capability. Delivery is governed by humans, but the pipeline as a whole is automated and orchestrated rather than manually sequential.

The gap between these stages is significant. **By 2028, teams that consistently apply AI across the entire software delivery lifecycle are forecast to achieve 25 - 30% productivity gains**. This is up substantially from only **10%** delivered by code-generation-focused approaches in 2024 (*Gartner, 2024*). The difference is not driven by tool sophistication. It is driven by process redesign.

For a technology leader evaluating their organisation's current state, the diagnostic question is critical. **If the AI coding tool were removed tomorrow, would delivery metrics change materially?** If the answer is no, the organisation is at Stage 1. The tool is optimising coding velocity without affecting overall delivery speed. If that is the case, it is worth asking whether the investment is in the right place. The real bottleneck may not be code generation. It may be requirements clarity, review capacity, test infrastructure, or approval governance. Investing in code generation speed whilst the pipeline bottleneck remains unaddressed is **funding the symptom rather than solving the underlying problem**.

Questions for Technology Leaders to Ask

The experience of organisations adopting AI tooling suggests a series of diagnostic questions that technology leaders should confront.

Has deployment frequency actually improved? Measure deployment frequency before and after AI tooling adoption. Is code reaching production faster, or is it sitting in the same release approval queues and test cycles as before, only arriving faster?

Can the organisation quantify investment against return? How much has been spent on AI tooling, training, and process change? What measurable efficiency gains and business value have been realised? Where those gains accrue (faster time to market, reduced cost per feature, improved software quality, reduced technical debt) matters because some organisations see faster code without corresponding improvements in the business metrics that matter.

Has the defect rate or security exposure improved? If code is being shipped faster, has quality improved, stayed constant, or declined? Has security exposure increased? In regulated sectors, this is the critical question; a faster delivery cadence with flat or deteriorating quality is a business risk, not a competitive advantage.

Do equivalent governance and integration tools match the code generation velocity? If code generation has accelerated but review, testing, and deployment automation have not, the organisation is likely shipping at unsustainable rates. Has governance matured alongside tooling velocity? Has architectural review kept pace? Is there a plan for technology retirement and technical debt reduction?

Where is the organisation in its AI adoption maturity? The three stages matter because they determine where the real returns are. Is the organisation at Stage 1 (individual tool), Stage 2 (team workflow), or Stage 3 (pipeline automation)? If at Stage 1 with flat delivery metrics, this signals that the real bottleneck is not code generation which is information that can be used to redirect investment.

Has the organisation adapted its SDLC for AI tooling? There is no point accelerating code development if the requirement gathering stage is failing to fuel the process or if the testing stage is a bottleneck. Every stage of the SDLC should be examined and adapted to maximise value from AI assistance.

Is there an AI maturity assessment in place? Organisations need clarity on their organisational and process readiness for AI. The deployment of AI tooling is not fundamentally a technical challenge; it is a business transformation challenge. An AI maturity assessment, anchored in business goals and the metrics to measure their achievement, is the foundation.



The Recommendation

The first step before deploying any AI tooling is to establish and document a comprehensive baseline of developer and delivery performance. This baseline must include DORA metrics (lead time for changes, deployment frequency, time to restore service, and change failure rate), individual developer productivity measures, code quality metrics, and security assessments. [This benchmarking discipline is non-negotiable](#); without it, post-adoption measurements cannot answer the fundamental question of whether the investment has delivered value. The baseline also serves a secondary but critical purpose which is establishing psychological clarity and expectations across the organisation about what will and will not change.

Technology leaders should measure pipeline throughput and business impact before and after any AI tooling decision. After implementation, those same metrics should be measured again at regular intervals to establish whether delivery has genuinely accelerated or whether code velocity has increased whilst overall delivery speed has remained flat.

Code generation speed and pipeline automation should be treated as two separate investments, because solving one does not solve the other. A faster code generator plugged into a sequential, bottlenecked pipeline will not deliver meaningful acceleration. The two must be addressed together. Organisations that move sequentially from [Stage 1](#) (individual tool) through [Stage 2](#) (team workflow) to [Stage 3](#) (pipeline automation) as

confidence grows and value is demonstrated see sustainable gains. Organisations that invest **only** in code generation without addressing pipeline constraints will likely see disappointing returns. If delivery metrics have not moved since adopting AI tools, the productivity gap is not a technology problem, it is a process and governance problem that should be funded and addressed accordingly. [The bottlenecks need examining empirically](#). Is it requirements clarity? Review capacity? Test infrastructure? Approval chains? Environment availability? Once the bottleneck is identified, it needs to be addressed directly which may require investment in tooling, training, process redesign, or organisational restructuring. Leaving bottlenecks in place whilst accelerating upstream activity simply creates friction.

The technology leaders who will deliver business value from AI in engineering delivery are those who measure the full delivery pipeline, understand where time actually goes, and invest in removing the real constraints. Code generation velocity is one component of that picture, but only one. The organisations achieving the highest returns on their AI investment are those who [treat AI adoption as an integrated business transformation challenge, not as a tooling rollout](#). They invest in capability building and process redesign alongside tools. They measure business impact, not just developer satisfaction, and they confront the hard questions about whether their deployment of AI is really solving the problem.

Organisations implementing the diagnostic framework outlined in this paper will benefit from a structured review of their current delivery pipeline against the three stages of AI adoption maturity. A tailored assessment can establish baseline metrics, identify specific bottlenecks, and determine which stage-appropriate interventions will deliver the greatest return on investment. **Abstract Group** provides such assessments, enabling technology leaders to translate the principles of this analysis into organisation-specific delivery improvement programmes.



Reference List

Atlassian (2025) State of Developer Experience Report. Available at: [State of Developer Experience Report 2025 | Atlassian](#)

CIO.com/Forrester (2024) Companies to shift AI goals in 2025 with setbacks inevitable. Available at: [Companies to shift AI goals in 2025 – with setbacks inevitable, Forrester predicts | CIO](#)

Deloitte (2026) State of AI in Enterprise. Available at: [The State of AI in the Enterprise - 2026 AI report | Deloitte Global](#)

DORA (2025) DevOps Research and Assessment Report. Available at: <https://dora.dev>

Gartner (2025) AI in Software Engineering. Available at: [Don't Limit AI in Software Engineering to Coding | Gartner](#)

McKinsey & Company (2025) Unlocking the value of AI in software development. [Available at: Leading AI-driven software organizations show the way | McKinsey](#)

Sonar (2026) Sonar Data Reveals Critical Verification Gap in AI Coding. Available at: [Sonar Data Reveals Critical “Verification Gap” in AI Coding: 96% Don’t Fully Trust Output, Yet Only...](#)

Book Your AI Readiness Assessment

Get in Touch

abstract-group.com